Summary

Introduction to Hive

In this module, you began by learning about OLTP and OLAP systems. Then you learnt about the key features of Hive and the Hive architecture. In addition, you went through some practical use cases of Hive, and learnt about the differences between an RDBMS and Hive.

Introduction to Hive

Hive was introduced to enable analysts to easily write high-level programming languages in Java or Python to execute MapReduce programs. Hive uses an SQL-like language called Hive Query Language (HQL), which makes it easy to execute MapReduce tasks on the HDFS.

OLTP vs OLAP

OLTP System	**OLAP System**
● **On-Line Transaction Processing** systems are those systems that support online transactions in databases on a real-time basis. The primary objective of these systems is to process data, not analyse it.	● **On-Line Analytical Processing** systems are those categories of software that provide analytical platforms to derive meaningful insights for business purposes.
● **Transactional**: OLTP refers to transactional data processing, and this mainly includes writing into databases.	● **Analytical**: OLAP systems are used for analytical purposes, wherein you perform analytical tasks to derive meaningful insights from the given data. OLAP is more of an offline data store where you perform analytics on a data set.
● **CRUD Operations**: OLTP systems support CRUD operations with which you can create, retrieve, update or delete data from a database.	● **Aggregation Operation**: OLAP systems are not meant to perform CRUD operations, because OLAP does not alter the existing database. You can only perform data aggregation on existing data sets to derive meaningful insights.
● **Short Query Time**: It should take less time to run a query using an OLTP system, as it needs to work fast in order to process transactions on a real-time basis.	● **Long Query Time**: OLAP systems have longer query times, as they need to process big data simultaneously at a particular point in time.
● **Mostly Normalised Data**: An OLTP system mostly contains the normalised form of data so	● **Mostly Denormalised Data:** The data present on an OLAP system is mostly denormalised and is available only in a single table, so that the

that the CRUD operations can be performed easily.	required inferences can be drawn from that table.
● **Absolute Data Integrity**: Operations are performed on a transactional basis and, hence, there has to be absolute data integrity and the tables should have a well-defined structure.	● OLAP systems do not maintain absolute data integrity.
● **Examples**: - Bank transactions - Online flight ticket booking system on Paytm - Adding items to the cart on the Amazon app	● **Examples**: - An e-commerce company wants to analyse its sales and products in order to build suitable marketing strategies. - The Government of India wants to conduct a demographic analysis of Aadhar card data to know the sex ratio in a particular state.

Note that OLTP systems are the original sources of data, whereas in the case of OLAP systems, data comes from different OLTP databases.

Key Features of Hive

What is Hive?

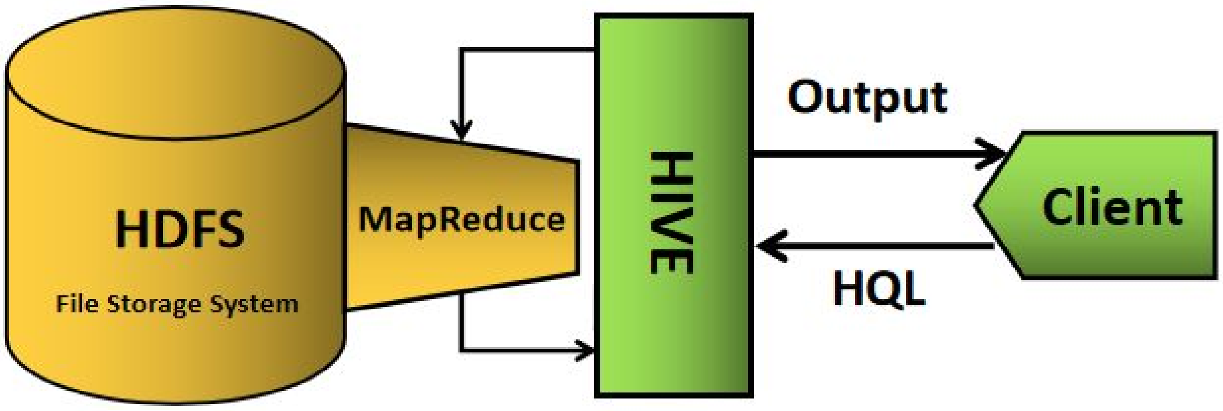
- **A software tool:** Hive is a software tool that is used to process the data available in the HDFS. It sits on top of the Hadoop system to summarise big data, and makes it easy to query and perform analysis.
- **SQL-like syntaxes:** Hive supports simple SQL-like syntaxes to query into the HDFS.
- **A lense between the HDFS and MapReduce:** Hive works as the lense between the HDFS and MapReduce; you write queries in Hive, and, ultimately, it triggers to the MapReduce job to convert the queries into MapReduce codes, which then operate on the data available in the HDFS.
- **Multiple storage formats:** An interesting aspect of Hive is that it allows you to fetch or query into any type of file format available in the HDFS.

What is Hive not?

- **An analytical tool:** Hive only supports OLAP systems. You can use Hive only for analytical purposes. It does not support real-time transactions in a database.
- **Generally no record-level updates:** Hive generally allows you to only append data to a database; it does not support record-level updates such as delete, update or alter.
- **Limited by SQL syntaxes:** Since Hive supports only simple SQL-like syntaxes, you cannot perform analyses that are too complex.

Features of Hive

- **Hive works on the HDFS:** Clients interact with the HDFS using Hive tools and use the Hive Query Language to query into the HDFS, which ultimately initiates a MapReduce job to fetch the data. Note that Hive is not the storage system; it is merely a software tool that works on top of the HDFS to query into the file storage system.



- **Single Write, Multi-Read:** As you learnt, Hive is an OLAP system that is designed to read data for analytical purposes. Hence, you can read the data as many times as you want, but you can perform write operations into the HDFS only once.
- **Hive Query Language:** Hive supports the Hive Query Language (HQL) for querying into the HDFS, although the HQL is ultimately translated into MapReduce jobs internally. Hence, internally, you have MapReduce and the HDFS even if you use Hive.
- **Metadata:** Hive fetches data from the HDFS in the form of tables and stores the schema of the data in its metastore. Metadata is data or information about some given data.
- **Tabular Structure:** In Hive, you read the data into tables after creating their schema using HQL. There are two types of tables in Hive, namely, internal and external.
- **Aggregation Functions**: Hive is an OLAP system that is used for analytical purposes. Hence, it has a variety of aggregation functions such as sorting, average, sum, count, min or max.
- **User-Defined Functions (UDFs):** Hive supports one of the dynamic features of user-defined functions. In addition to the in-built functions, Hive allows you to customise functions to process records and groups of records. In many industries, it is necessary to perform various tasks using UDFs, which are more extensible than the existing functions.
- **Batch Processing:** Consider a ride-hailing company such as Uber, which needs to process, at the end of each day, the data of all the rides availed and distribute the revenue among the drivers accordingly. Now, this process is not done immediately after each ride but at the end of each day. This is an example of batch processing.

  Batch processing in Hadoop refers to processing a group of transactions that have been gathered over a particular period of time. As in the aforementioned example, the particular period of time is one day. Multiple rides are availed in one whole day, with money being transacted between riders and drivers/Uber, and at the end of the day, the money thus received from the day's transactions is allocated to the drivers.

- **Highly Extensible:** Hive is highly extensible. If a particular application is written in Java or Python, it can interact with Hive and can fetch relevant data from the HDFS.

Use Cases of Hive

Some examples of use cases of Hive are as follows:

- **Reporting:** Consider a mutual funds company where the average, minimum or maximum price of stocks needs to be calculated at the end of each day. This process is called reporting, wherein you need to run scheduled reports in a particular period of time.

- **Ad-hoc Analysis:** Let us take the example of the mutual funds company again, except this time, you are not supposed to present the report of stock price at the end of each day. Instead, you want to know the current price of a group of shares and predict the price of the stocks after one hour, based on the fall or rise in the share price. This is called an ad hoc analysis, wherein you query into a database for a particular ad-hoc demand.

- **Machine Learning:** As you have learnt already, Hive is an OLAP system, wherein you conduct data analysis and build machine learning algorithms to derive meaningful insights from data.

Basic Architecture of Hive

The Hive ecosystem is divided into the following parts:

1. **Hive Client:** The Hive Client acts as the interface for the Hive system through which users interact with Hive and query into the HDFS. It consists of the following parts:
   a. **Command-Line Interface (CLI):** This is an interface between Hive and its users. You need to write queries in HQL to derive results from the data available in the HDFS. The CLI acts as the command-line tool for the Hive server.
   b. **Web Interface:** This is the graphical user interface of Hive. It is an alternative to the Hive CLI, wherein you query into the HDFS through the Hive tool.

2. **Hive Core:** The Hive Core is the core part of the Hive architecture, which links the Hive client and the Hadoop cluster. It consists of the following three parts:
   a. **Hive QL Process Engine:** When you write a query in HQL via the CLI or web interface, it goes into the Hive QL Process Engine, which checks the syntax of the query and analyses the same. The HQL is similar to SQL for querying, and it uses the schema information available in the metastore.

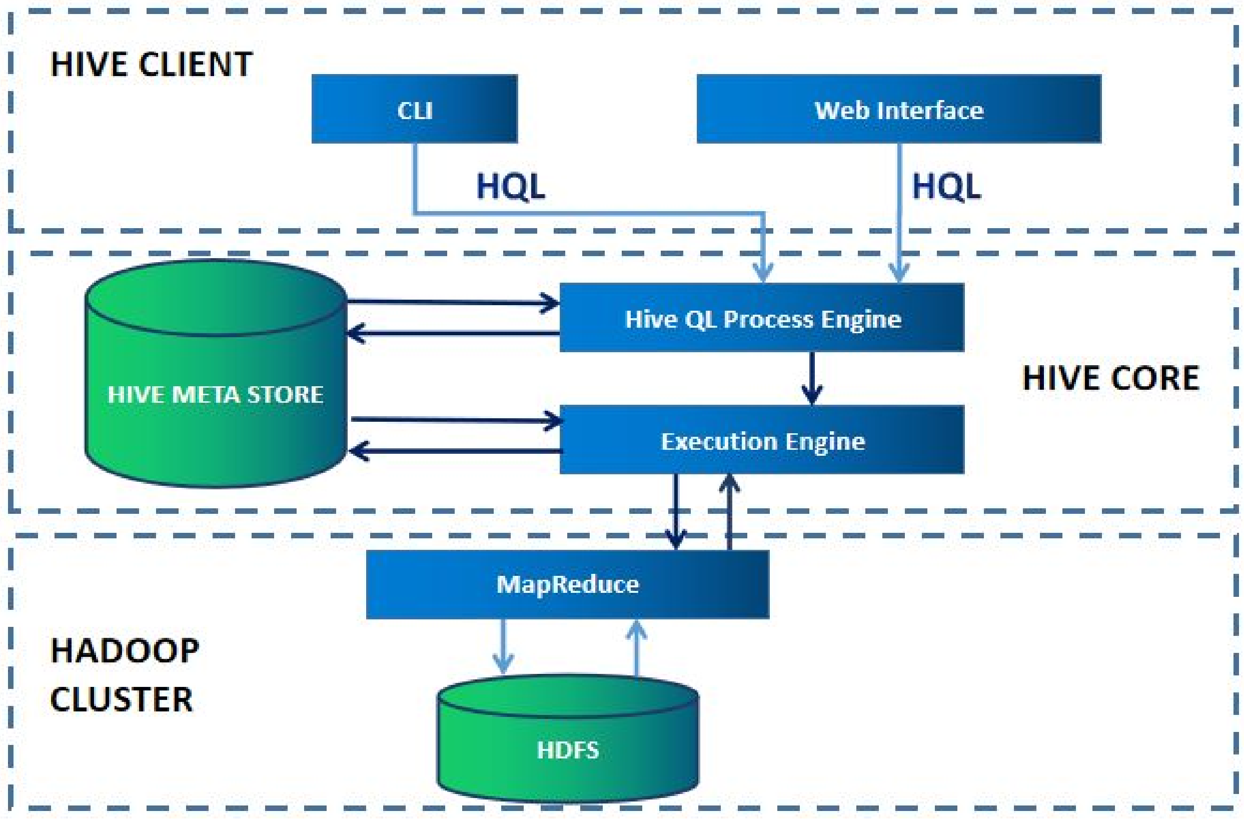
   b. **Hive Metastore:** The metastore can be considered the heart of the Hive ecosystem. Let us try to understand this with the help of an example. Suppose an e-commerce company has to manage its employees, customers, orders, etc. Data about these is available in the HDFS, and you want to perform analytics on this data.

The information about the tables and the database is called the metadata.

It is important to note that Hive is not a database, and the metastore acts as a storage space for Hive, which stores the schema information of all the tables that have been created. The metastore does not store the actual data; instead, it stores the metadata of the tables that have been created. It contains its own embedded RDBMS to store the relational tables and uses Apache's 'Derby' RDBMS for this purpose.

c. **Execution Engine:** The execution engine is used to convert Hive queries to MapReduce jobs. The engine processes the queries and generates results that match the MapReduce results.

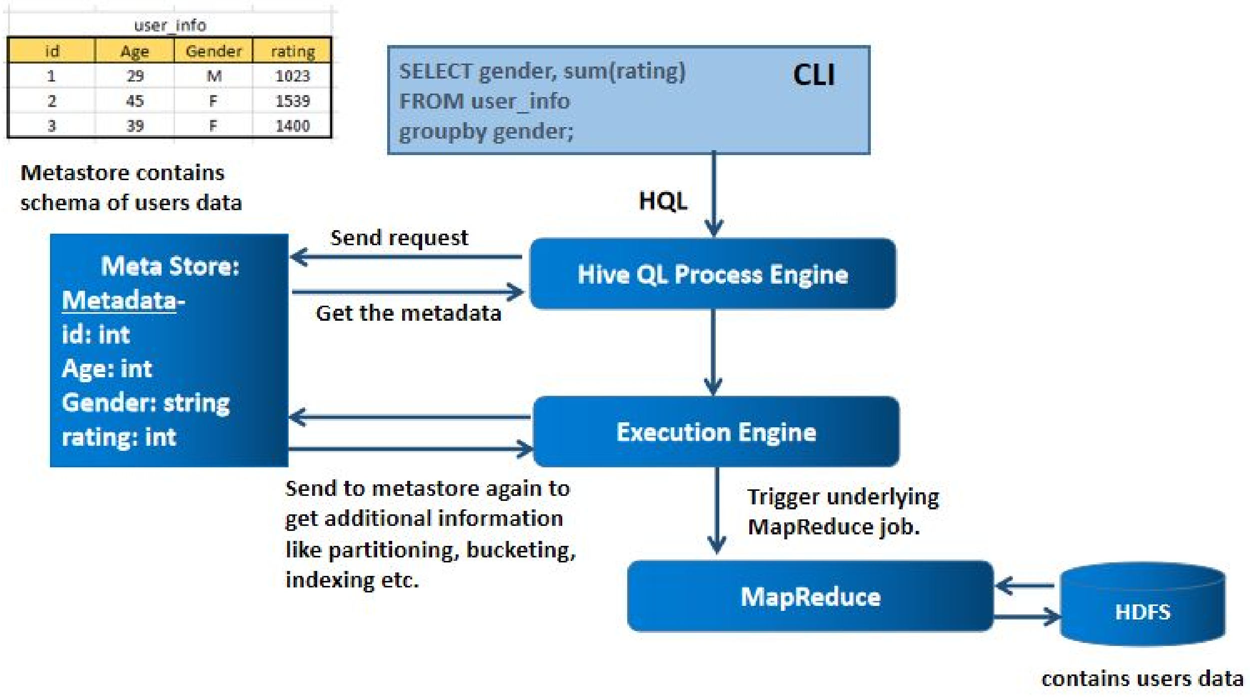
3. **Hadoop Cluster:** The Hadoop Cluster is the file storage system where users can store files in either a structured or an unstructured format, and can query into the HDFS using MapReduce.



The Hadoop Cluster is the bottom-most block of the Hive architecture. It consists of the following two parts:

- **HDFS:** The Hadoop Distributed File System (HDFS) allows you to store large volumes of data across multiple nodes in a Hadoop cluster. It is used to perform distributed computing on structured or unstructured data.
- **MapReduce:** This is a programming framework that is used by Hadoop for processing data. After users store data in the HDFS, MapReduce is used to query into the HDFS to perform data analysis. Note that even if you are running a Hive job, a MapReduce job gets triggered in the background to query into the HDFS.

In the example below, you can see how a Hive query is executed and how it triggers a MapReduce job.



id	Age	Gender	rating
1	29	M	1023
2	45	F	1539
3	39	F	1400

```
SELECT gender, sum(rating)
FROM user_info
groupby gender;
```

upGrad

Hive vs Relational Databases

Hive	RDBMS
● Hive works on '**Schema on READ**'. ● In Hive, the entire data dump is available in the HDFS in a raw format. In order to read the data from the HDFS, you need to create a schema. ● Hive is meant for OLAP purposes and so, you need to create a schema every time you want to read the data stored in the HDFS. This schema is stored in the Hive metastore. Remember, Hive is not a database, as data is stored only in the HDFS, whereas Hive only stores the schema information of the tables that are defined by the user.	● An RDBMS works on '**Schema on WRITE**'. ● In an RDBMS, you need to define the schema of the tables before storing the data in the database. Hence, it is called 'schema on write'.
● Hive is meant for OLAP systems; hence, record level updates are generally not possible in Hive. Also, Hive only supports appending the data, whereas the RDBMS supports functions such as alters, delete, drop, etc.	● An RDBMS supports record-level or row-level updates.
● Hive is used to deal with petabytes of data.	● Deals with Terabytes of data
● Due to the large data size, Hive takes longer to compute. ● Another reason for the long computation time is that Hive supports 'schema on read', which means every time you want to read the data, you need to match it with the schema.	● In an RDBMS, data is stored on the basis of the predefined schema; hence, computation takes less time.

upGrad

Summary

Basic Hive Queries

In this session, you learnt about some of the common Hive syntaxes and query optimisation techniques.

Database Creation

All Hive demonstrations were performed on an EMR cluster (Launching and Connecting to the EMR Cluster).

Since you have worked with a data set in an S3 bucket, here are the ways to load data from S3:

1. You can load the data stored in S3 into Hive directly through the S3 link. However, any changes made to the bucket will cause changes while loading the data in Hive.
2. Another method could be to copy the data from S3 into the cluster and then load it into the Hive tables. This will help you keep your data safe within the cluster.

The commonly used queries for creating databases are as follows:

- You can create a database with the following query:
  create database if not exists demo ;.
  This database named 'demo' is created in the Hive warehouse itself.
- To print the description of the database, you can write a query with the keyword "**extended**":
  describe database extended demo2 ;.
- If you want to see the database's headers, then you will need to set the headers to true:
  set hive.cli.print.header=true ;.

Internal and External Tables I

There are two types of tables in Hive:

- Internal tables
- External tables

The properties of internal and external tables are described below:

- **Creation of tables:** Any table in Hive is an internal table by default. To create an external table, you need to write the keyword “EXTERNAL” explicitly while writing the query to create the table.

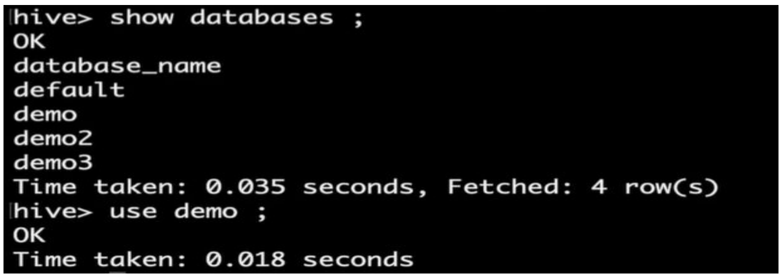
- **Internal table vs external table:** The basic difference between internal and external tables is that when you create an internal table, Hive is itself responsible for the data, and it takes control of the entire life cycle of the metadata as well as the actual data itself.

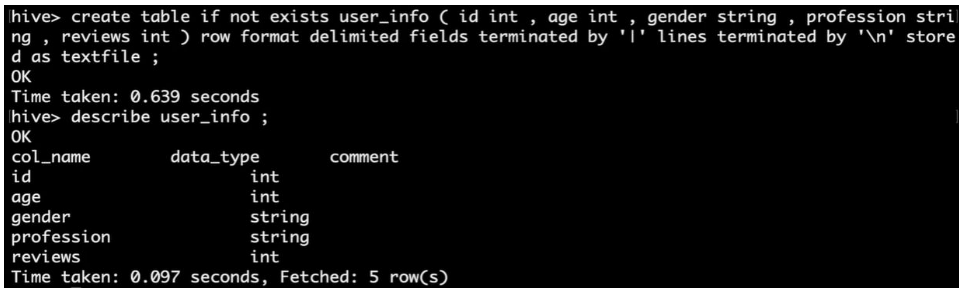
  In contrast, when you create an external table, Hive is responsible for taking care of the table schema only. It does not take care of the actual data.

- When you apply the drop command on an internal table, the entire metadata available in the metastore, as well as the actual table data that is present in the HDFS, gets deleted.

  However, when you apply the drop command on an external table, the entire metadata available in the metastore gets deleted, but the actual table data available in the HDFS remains intact there.

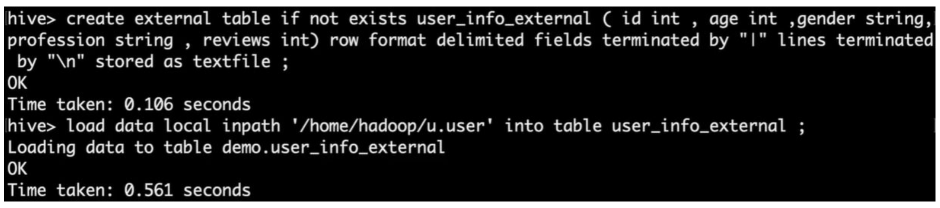
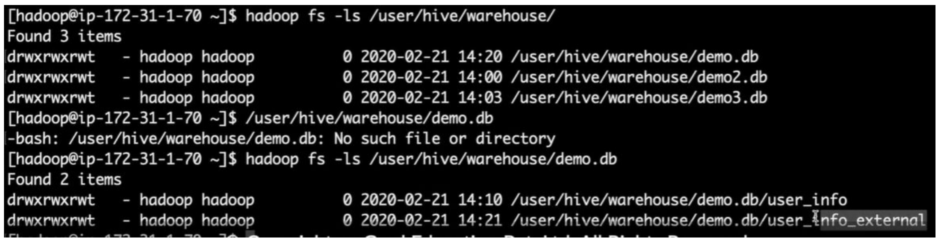
- **When to use INT/EXT tables:** Suppose there is a particular table that is very common and can be used by multiple applications. In such a case, it is preferable to use an external table, because when you drop the table, the data would remain intact in the HDFS and can be used by some other application. However, when you drop an internal table, the schema as well as the table data get deleted, and they become unavailable for other applications as well.

Database selection	
	`hive> show databases ;`
	`OK`
	`database_name`
	`default`
	`demo`
	`demo2`
	`demo3`
	`Time taken: 0.035 seconds, Fetched: 4 row(s)`
	`hive> use demo ;`
	`OK`
	`Time taken: 0.018 seconds`

Creating an internal table	
	`hive> create table if not exists user_info ( id int , age int , gender string , profession stri`
	`ng , reviews int ) row format delimited fields terminated by '|' lines terminated by '\n' store`
	`d as textfile ;`
	`OK`
	`Time taken: 0.639 seconds`
	`hive> describe user_info ;`
	`OK`
	`col_name        data_type       comment`
	`id                      int`
	`age                     int`
	`gender                  string`
	`profession              string`
	`reviews                 int`
	`Time taken: 0.097 seconds, Fetched: 5 row(s)`
	
	By using the query above, you are also specifying that in the data available in the HDFS, fields are delimited by '|' and lines are terminated by '\n'.

Loading data into the tables	

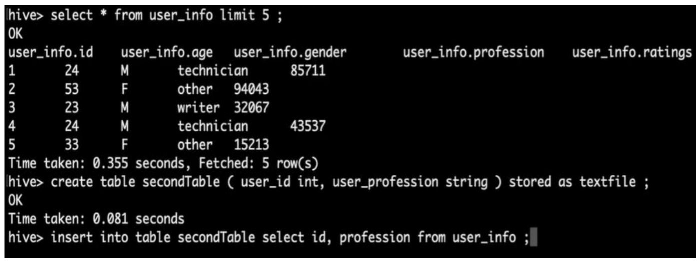
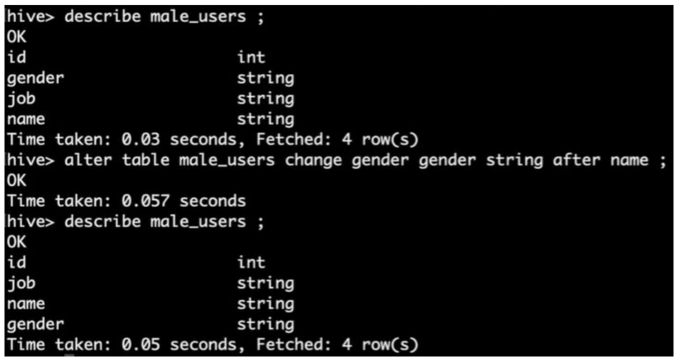
Internal and External Tables II

Creating an external table	
Location of the external table	

```
hive> create external table if not exists user_info_external ( id int , age int ,gender string,
profession string , reviews int) row format delimited fields terminated by "|" lines terminated
 by "\n" stored as textfile ;
OK
Time taken: 0.106 seconds
hive> load data local inpath '/home/hadoop/u.user' into table user_info_external ;
Loading data to table demo.user_info_external
OK
Time taken: 0.561 seconds
```

```
[hadoop@ip-172-31-1-70 ~]$ hadoop fs -ls /user/hive/warehouse/
Found 3 items
drwxrwxrwt   - hadoop hadoop          0 2020-02-21 14:20 /user/hive/warehouse/demo.db
drwxrwxrwt   - hadoop hadoop          0 2020-02-21 14:00 /user/hive/warehouse/demo2.db
drwxrwxrwt   - hadoop hadoop          0 2020-02-21 14:03 /user/hive/warehouse/demo3.db
[hadoop@ip-172-31-1-70 ~]$ /user/hive/warehouse/demo.db
-bash: /user/hive/warehouse/demo.db: No such file or directory
[hadoop@ip-172-31-1-70 ~]$ hadoop fs -ls /user/hive/warehouse/demo.db
Found 2 items
drwxrwxrwt   - hadoop hadoop          0 2020-02-21 14:10 /user/hive/warehouse/demo.db/user_info
drwxrwxrwt   - hadoop hadoop          0 2020-02-21 14:21 /user/hive/warehouse/demo.db/user_info_external
```

Operations on Tables

Inserting data from an existing table to a newly created table	
Changing the position of the gender column from second to last by using the "**alter**" command	

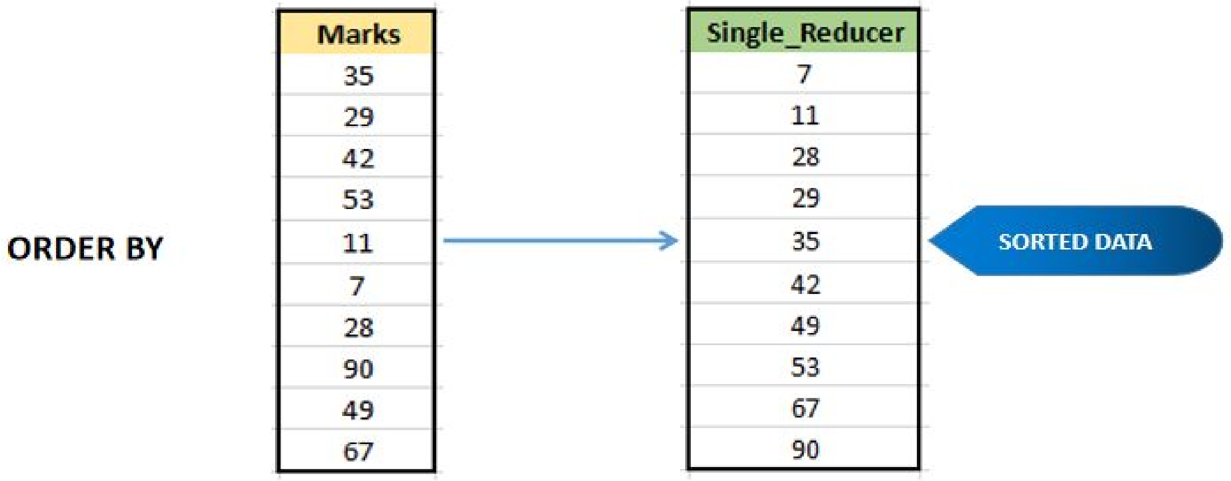
```
hive> select * from user_info limit 5 ;
OK
user_info.id    user_info.age   user_info.gender        user_info.profession    user_info.ratings
1       24      M       technician      85711
2       53      F       other   94043
3       23      M       writer  32067
4       24      M       technician      43537
5       33      F       other   15213
Time taken: 0.355 seconds, Fetched: 5 row(s)
hive> create table secondTable ( user_id int, user_profession string ) stored as textfile ;
OK
Time taken: 0.081 seconds
hive> insert into table secondTable select id, profession from user_info ;
```

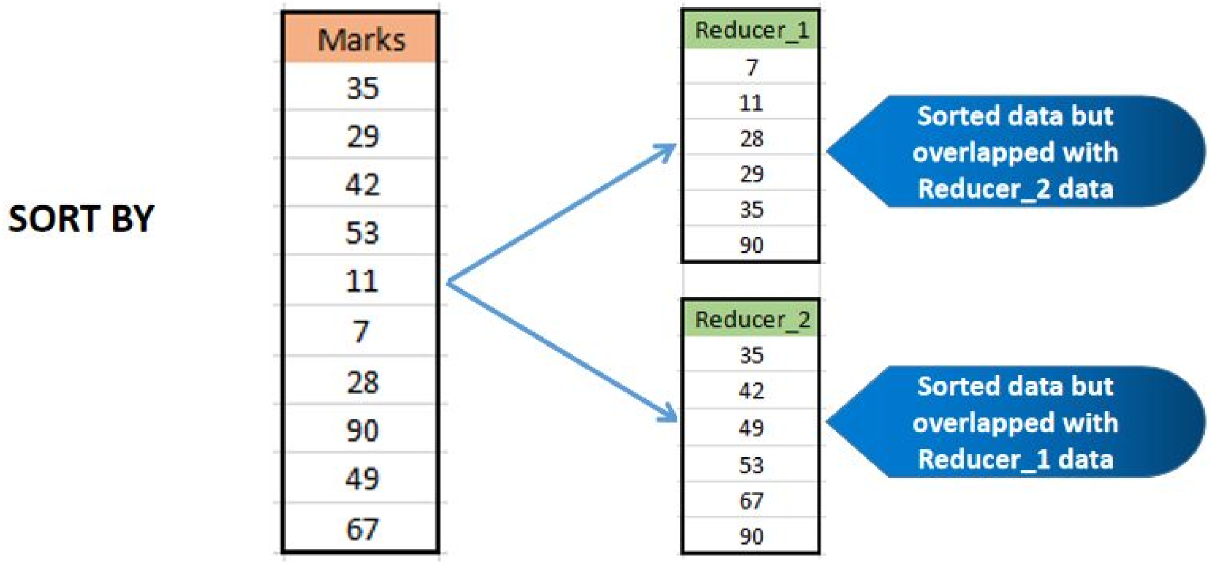
```
hive> describe male_users ;
OK
id                      int
gender                  string
job                     string
name                    string
Time taken: 0.03 seconds, Fetched: 4 row(s)
hive> alter table male_users change gender gender string after name ;
OK
Time taken: 0.057 seconds
hive> describe male_users ;
OK
id                      int
job                     string
name                    string
gender                  string
Time taken: 0.05 seconds, Fetched: 4 row(s)
```

Order By and Sort By

The “ORDER BY” clause gives a complete sorted list of the data and guarantees total ordering in the output. It may use multiple mappers but uses only one reducer for this purpose.

The “SORT BY” clause sorts the data at the reducer level only. It does not guarantee total ordering of the data as performed by the “ORDER BY” clause. This command does not give a complete sorted list. Also, there may be overlapping data in the two reducers.



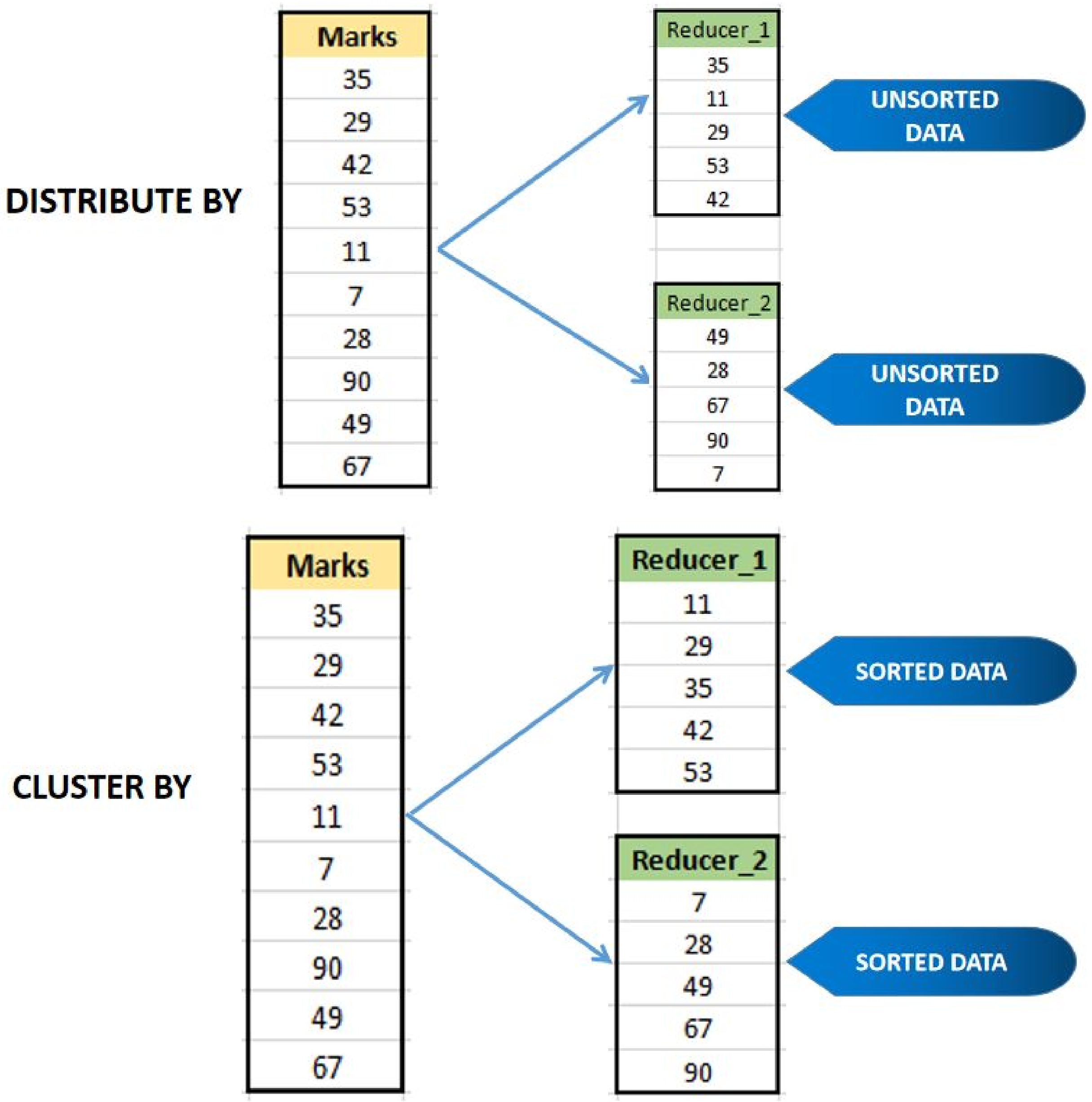

SORT BY
Marks
35
29
42
53
11
7
28
90
49
67
Reducer_1
7
11
28
29
35
90
Sorted data but overlapped with Reducer_2 data
Reducer_2
35
42
49
53
67
90
Sorted data but overlapped with Reducer_1 data

Distribute By and Cluster By

The “DISTRIBUTE BY” clause is used to distribute data among multiple reducers. The data does not get sorted into each reducer; it is merey distributed among reducers using this command.

On the other hand, with the “CLUSTER BY” clause, data is distributed among multiple reducers, with each reducer containing sorted and non-overlapping data.

Thus, you can conclude that the “CLUSTER BY” clause is a combination of the “DISTRIBUTE BY” and “SORT BY” clauses.

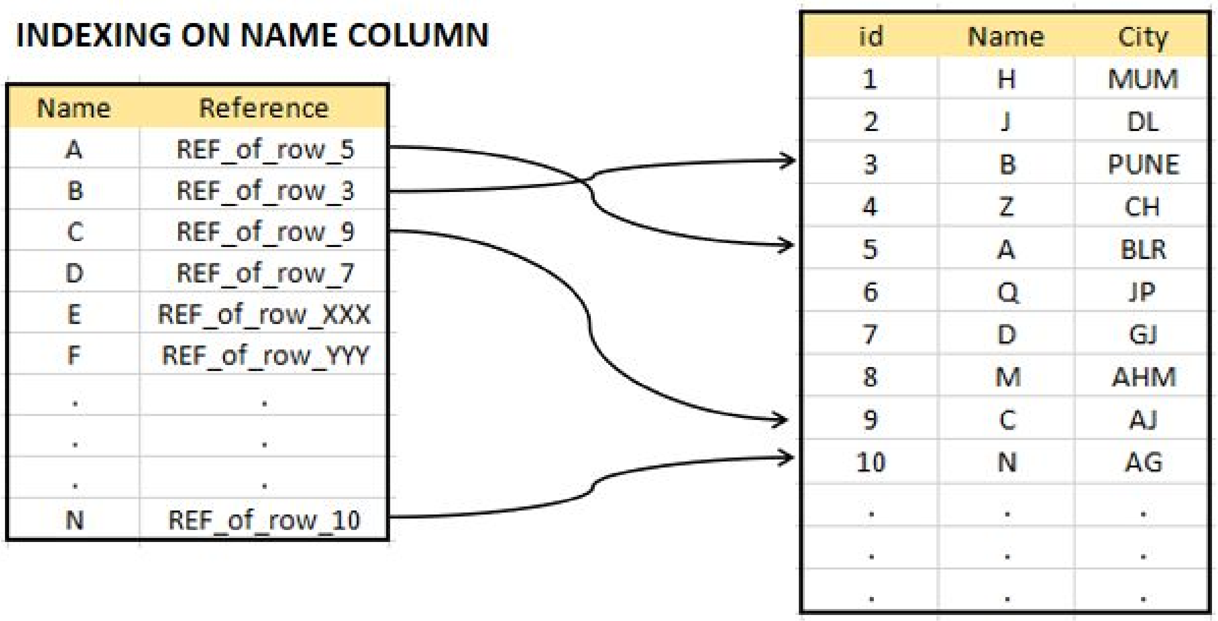


User Defined Functions

User-defined functions can be created through the following steps:

- Create a Java program
- Save the Java file as a 'jar' file
- Add the 'jar' file to Hive. Use the following command: **add jar /home/hadoop/udfDemo.jar ;**.
- Create a function of the 'jar' file that you added
- Use those function in the Hive query

Indexing I



Name	Reference
A	REF_of_row_5
B	REF_of_row_3
C	REF_of_row_9
D	REF_of_row_7
E	REF_of_row_XXX
F	REF_of_row_YYY
.	.
.	.
.	.
N	REF_of_row_10

id	Name	City
1	H	MUM
2	J	DL
3	B	PUNE
4	Z	CH
5	A	BLR
6	Q	JP
7	D	GJ
8	M	AHM
9	C	AJ
10	N	AG
.	.	.
.	.	.
.	.	.

As you can see in the image above, indexing has been done on the Name column. Each name has been added in the sorted order to the separate data structure, and there is a reference corresponding to each name, which represents the rows in the original table.

So, when you search for a particular name, it goes to the separate data structure and applies the 'binary search algorithm', and then returns the address of the row in the original table and gives the results.

There are two types of indexing in Hive:

- Compact indexing
- Bitmap indexing

You can refer to the following link to learn more about the two types of indexing:

[Types of Indexing]

You can use indexing in the following scenarios:

- When the data set is very large
- If you need a short query execution time
- While applying indexing on columns that are used more frequently than others
- For read-heavy applications where you need to read data more frequently.

Generally it is not preferable to use indexing in writing heavy applications because each time when new data will come, internally all the new updates have to be done in the separate data structure also. Simultaneously, a separate data structure has to remain in the sorted state each time when you add new data in it, which is a time consuming task to edit that data structure each time while writing the data in the actual table.

Indexing II

- **COMPACT Keyword:** Using this keyword, you can specify the type of indexing that you are going to use for a particular column. For bitmap indexing, you need to use the BITMAP keyword.
- **Deferred Rebuild:** It is used to defer the process of indexing to a future time point, which means indexing is not active right now. To activate indexing, you need to write a separate query as follows:
  alter index i1 on user_info rebuild ;

Summary

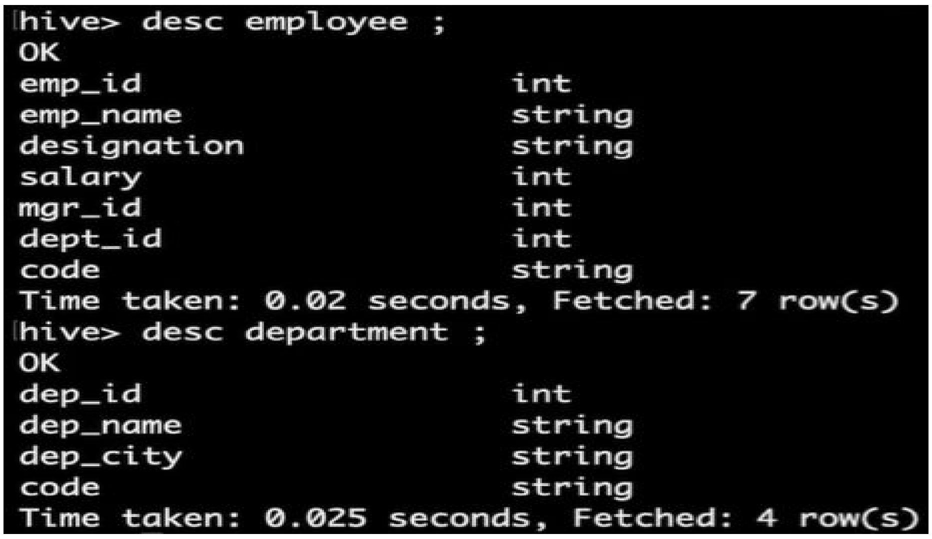
Advanced Hive Queries

In this session, you learnt about advanced Hive queries. These concepts included map joins, partitioning, bucketing, etc.

Joins in Hive

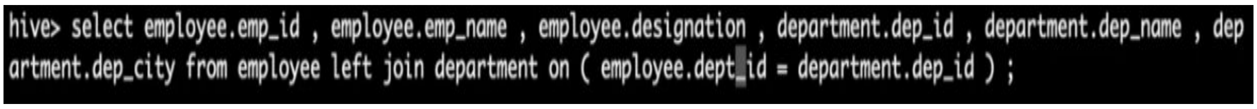
In the case of joins both mapper and reducer works simultaneously. If any how you can remove the reducer from the operation and then perform the join using only mapper, you can decrease the query time. This is called “Map Join”.

Suppose, there are two tables named “employee” and “department”. Both of the tables contain columns related to employees and department respectively.



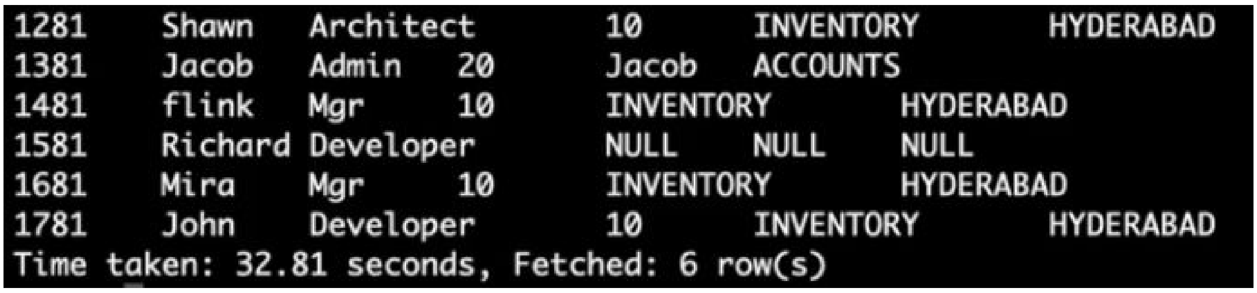
```
hive> desc employee ;
OK
emp_id                  int
emp_name                string
designation             string
salary                  int
mgr_id                  int
dept_id                 int
code                    string
Time taken: 0.02 seconds, Fetched: 7 row(s)
hive> desc department ;
OK
dep_id                  int
dep_name                string
dep_city                string
code                    string
Time taken: 0.025 seconds, Fetched: 4 row(s)
```

Using the query given below, we have performed a left join on “employee” and “department” tables.



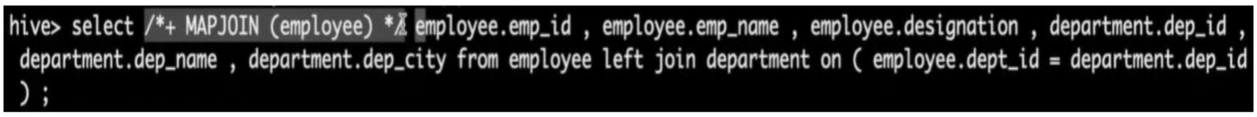
```
hive> select employee.emp_id , employee.emp_name , employee.designation , department.dep_id , department.dep_name , dep
artment.dep_city from employee left join department on ( employee.dept_id = department.dep_id ) ;
```

On executing the query, we get the following result.



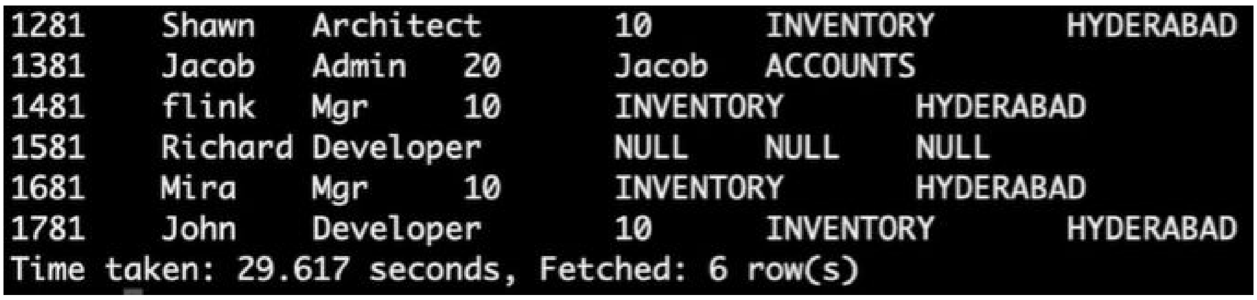
```
1281    Shawn   Architect       10      INVENTORY       HYDERABAD
1381    Jacob   Admin   20      Jacob   ACCOUNTS
1481    flink   Mgr     10      INVENTORY       HYDERABAD
1581    Richard Developer       NULL    NULL    NULL
1681    Mira    Mgr     10      INVENTORY       HYDERABAD
1781    John    Developer       10      INVENTORY       HYDERABAD
Time taken: 32.81 seconds, Fetched: 6 row(s)
```

Using a Map join, you can remove the reducer from the join operation and then perform the operation using the mapper only, thus reducing the query time. To perform a map join, you need to specify the hint to the above join query as shown below:



```
hive> select /*+ MAPJOIN (employee) */ employee.emp_id , employee.emp_name , employee.designation , department.dep_id ,
 department.dep_name , department.dep_city from employee left join department on ( employee.dept_id = department.dep_id
 ) ;
```

Without the map join, the query took 32.81 seconds to execute but after using Map Joins, the time was reduced to 29.617 seconds.

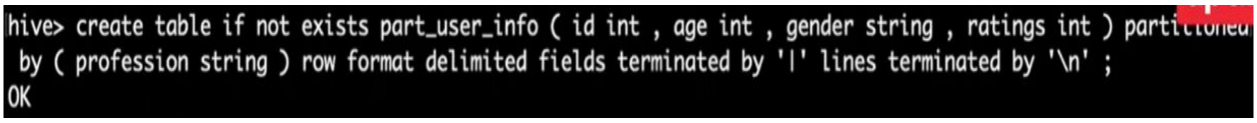


```
1281    Shawn   Architect       10      INVENTORY       HYDERABAD
1381    Jacob   Admin   20      Jacob   ACCOUNTS
1481    flink   Mgr     10      INVENTORY       HYDERABAD
1581    Richard Developer       NULL    NULL    NULL
1681    Mira    Mgr     10      INVENTORY       HYDERABAD
1781    John    Developer       10      INVENTORY       HYDERABAD
Time taken: 29.617 seconds, Fetched: 6 row(s)
```

Static Partitioning

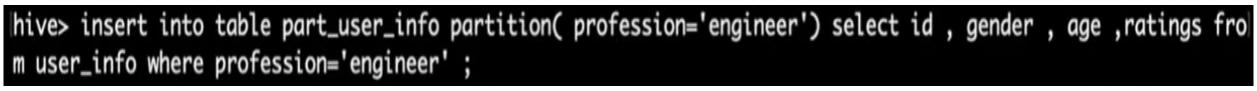
There are two types of partitioning in Hive:
- **Static partitioning:** In static partitioning, you need to load the data manually into the partitions.
- **Dynamic partitioning:** In dynamic partitioning, data gets allocated to the partitions automatically.

The syntax to apply static partitioning to a table is as follows:



```
hive> create table if not exists part_user_info ( id int , age int , gender string , ratings int ) partitioned
 by ( profession string ) row format delimited fields terminated by '|' lines terminated by '\n' ;
OK
```

Now, the data related to profession equals to ‘engineer’ is added manually to the partitions:

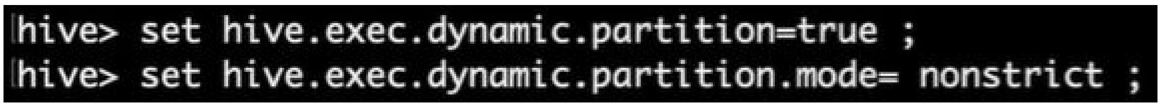


```
hive> insert into table part_user_info partition( profession='engineer') select id , gender , age ,ratings fro
m user_info where profession='engineer' ;
```

Dynamic Partitioning and Drop the Partitions

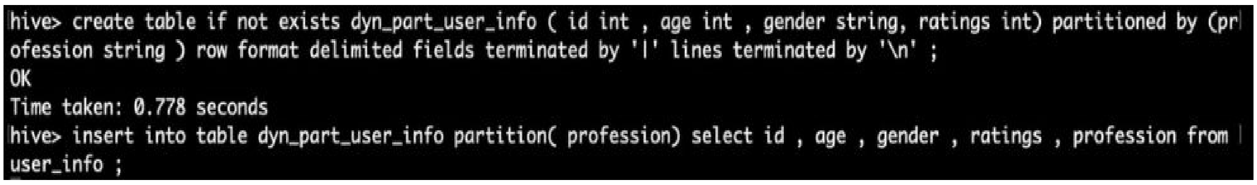
In dynamic partitioning, data gets allocated to the partitions automatically. The syntax to apply static partitioning to a table is as follows:

- By default, Hive does not allow dynamic partitioning; you need to define dynamic partitioning equals to true while writing the syntax:



```
hive> set hive.exec.dynamic.partition=true ;
hive> set hive.exec.dynamic.partition.mode= nonstrict ;
```

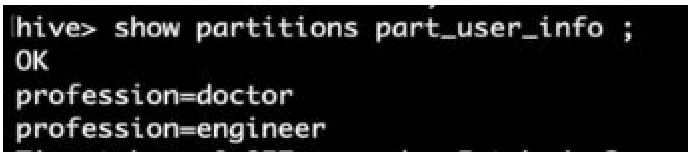
- In the next code snapshot, you can see how a dynamically partitioned table named "dyn_part_user_info" has been created and how data has been loaded into it using the "insert" clause:



```
hive> create table if not exists dyn_part_user_info ( id int , age int , gender string, ratings int) partitioned by (pr
ofession string ) row format delimited fields terminated by '|' lines terminated by '\n' ;
OK
Time taken: 0.778 seconds
hive> insert into table dyn_part_user_info partition( profession) select id , age , gender , ratings , profession from
user_info ;
```

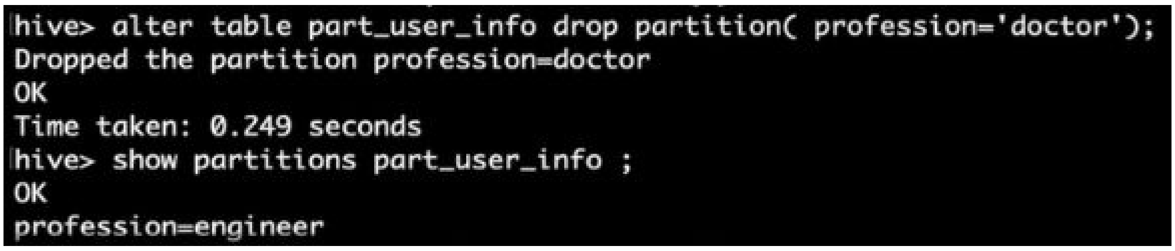
- Once you have loaded the data into the dynamically partitioned table "dyn_part_user_info", profession is allocated automatically to multiple partitions. These partitions are located in the "**user/hive/warehouse/dyn_part_user_info**" directory in Hadoop.

- You can view the partitions by running the following command:



```
hive> show partitions part_user_info ;
OK
profession=doctor
profession=engineer
```

- To drop the partition 'doctor', the command is as follows:



```
hive> alter table part_user_info drop partition( profession='doctor');
Dropped the partition profession=doctor
OK
Time taken: 0.249 seconds
hive> show partitions part_user_info ;
OK
profession=engineer
```

Partitioning is ideal when:

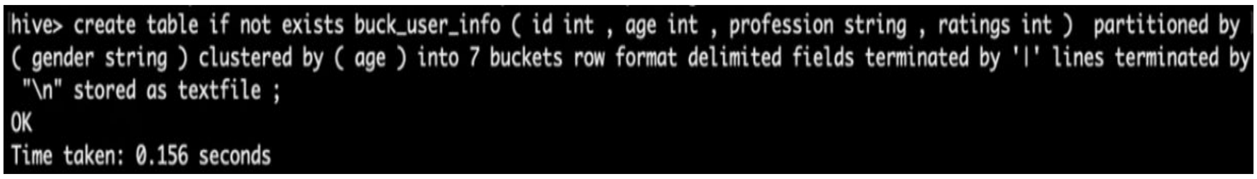
1. The sizes of the partitions are comparable to each other
2. The number of partitions is limited

Bucketing

Bucketing enables you to divide the data into partitions of equal size. It is preferred over Partitioning when the number of partitions are highly unbalanced or there are too many partitions resulting in over partitioning. In general practice, first you create the partitions and then perform bucketing.

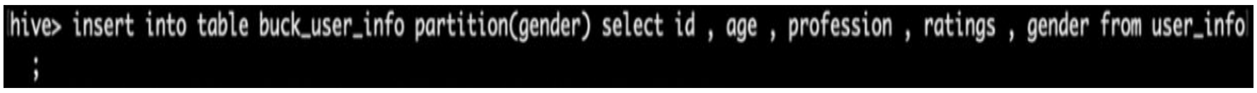
The syntax to apply bucketing is as follows:

- Initially, partitions have been created on the basis of gender. Then for each gender, seven buckets have been created on the basis of ‘age’ in the “buck_user_info” table:



```
hive> create table if not exists buck_user_info ( id int , age int , profession string , ratings int )  partitioned by
( gender string ) clustered by ( age ) into 7 buckets row format delimited fields terminated by '|' lines terminated by
 "\n" stored as textfile ;
OK
Time taken: 0.156 seconds
```

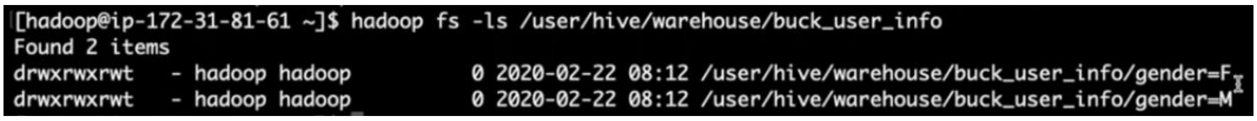
- Once the tables have been created, the data is loaded into it. In this query, the data is copied from the table “user_info” to the table “buck_user_info”:



```
hive> insert into table buck_user_info partition(gender) select id , age , profession , ratings , gender from user_info
 ;
```

- Now, let us revise the memory locations in Hadoop where these buckets are created:

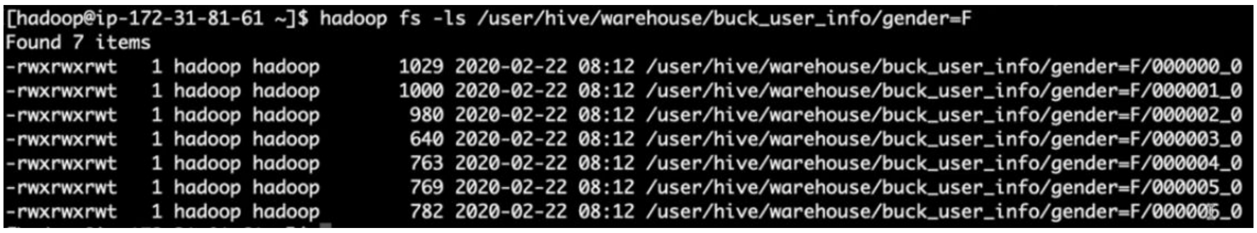
1. Let us first see the partitions created on the basis of gender:



```
[hadoop@ip-172-31-81-61 ~]$ hadoop fs -ls /user/hive/warehouse/buck_user_info
Found 2 items
drwxrwxrwt   - hadoop hadoop          0 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F
drwxrwxrwt   - hadoop hadoop          0 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=M
```

   So, there are two partitions that have been created on the basis of the genders ‘M’ and ‘F’.

2. To check the memory locations of the buckets for the partitions of the female gender, use the following commands:



```
[hadoop@ip-172-31-81-61 ~]$ hadoop fs -ls /user/hive/warehouse/buck_user_info/gender=F
Found 7 items
-rwxrwxrwt   1 hadoop hadoop       1029 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000000_0
-rwxrwxrwt   1 hadoop hadoop       1000 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000001_0
-rwxrwxrwt   1 hadoop hadoop        980 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000002_0
-rwxrwxrwt   1 hadoop hadoop        640 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000003_0
-rwxrwxrwt   1 hadoop hadoop        763 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000004_0
-rwxrwxrwt   1 hadoop hadoop        769 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000005_0
-rwxrwxrwt   1 hadoop hadoop        782 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000006_0
```

   You can see that seven files have been created in the folder containing the partitions for the female gender. These files correspond to the bucketing on the basis of age.

One key difference between bucketing and partitioning is that when you create partitions, new folders for each partition are created, but in the case of bucketing, multiple files (buckets) are created inside a single folder.